NASA Cost Estimating Symposium

Flight Software Cost Growth: Analysis and Recommendations

Dr. Jairus Hihn
Dr. Hamid Habib-agahi

March 2, 2000

Mission and System Architecture Section
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109

AGENDA

- Methodology
- Sample Summary Information
- Cost Growth Sources
- Recommendations
- What's Next

Flight Software Cost Risk Study: Methodology

1) Identified 8 missions and 11 participants
2) Unstructured Interview based on Protocol Analysis
3) Identified initial risk categories
4) Follow up Structured Interview to verify risk categories and to identify additional information
5) Draft Flight Software Cost Risk Management Report on causes and recommendations
6) Workshop/Focus Group to brainstorm underlying causes of software cost growth and to review recommendations
7) Multi-voting to identify top strategic policy recommendations
8) Updated Flight Software Cost Risk Management Report
9) Second Workshop/Focus Group to develop JPL Policy Recommendations and SW Development Principles

Summary of Mission Characteristics

Mission	Flight/ Ground	In-house vs. Contract	Current Phase	Cost Growth > 20%	Number of Participants
Mission 1	Flight	In-house	Operations	Yes	2
Mission 2	Flight	In-house	Completed	Yes	2
Mission 3	Flight	In-house	Operations	Yes	1
Mission 4	Flight	Contract	Operations	Yes	1
Mission 5	Flight	Contract	Operations	Yes	1
Mission 6	Flight	Contract	Implementation	Yes	2
Mission 7	Ground	In-house	Implementation	No	1
Mission 8	Ground	In-house	Implementation	Yes	1

Software Cost Growth Summary

SW Cost Growth (Percent of SW Budget)	
Mean	Range
51%	25%-71%

Reported Risk Area Frequency with Summary Details

Risk Area	% of Missions Responses	Summary of Reported Issues
Experience & Teaming	71%	• Management and system engineers had extensive hardware experience but insufficient software experience • Weak teaming between hardware, software and systems teams • SW engineers lacked system and mission experience
Planning	71%	• Poor planning and estimation practices • Planned inheritance never happened • Insufficient reserves for SW
Requirements & Design	57%	• Lack of good system architecture and system partitioning • Lack of good software architecture • Systems decisions made without accounting for impact on software • SW requirements solidify late in the life cycle and are very volatile
Testing	71%	• Testbeds; too few, too late, not validated, insufficient capability
Software Inheritance	57%	• Inherited code did not behave as advertised, was poorly documented, and required more modification than expected. (5 of 8 missions attempted to inherit software. Of these, 4 reported major problems.)
Tools & Methods	86%	• Poor test result analysis tools • Purchased COTS tool never used.
Staffing	71%	• High turnover in software staff • SW team was not included in early stages of planning • Integration and SW teams were not available to support ATLO

Cost Risk Impact

RISK AREA	Range	Mean
Experience & Teaming	5-15%	10%
Planning	20-50%	35%
Requirements & Design	10-50%	25%
Staffing	5-25%	10%
Testing	10-30%	15%
Tools	5-20%	5%

Summary of Software Cost Growth Sources by Importance

Risk Area	Frequency of Occurrence (% of Projects)	Estimated Contribution To Cost Growth	Which risk areas do you think are the most important for JPL to address?	Which risk areas do you think are the most important for project managers to consider?
Planning (incl. Control)	71%	35%	28%	28%
Requirements & Design (incl. Architecture & SW volatility)	57%	25%	15%	13%
Experience & Teaming	71%	10%	20%	25%
Testing	71%	15%	14%	10%
Staffing	71%	10%	7%	10%
Software Inheritance	57%	Incl. In Planning	11%	3%
Tools/ Methods	86%	5%	5%	6%

Top Five Risk Areas: The Causes Flight Software Cost Growth

Risk Area	Cost Growth Sources	Cost Growth Causes		
		Process	**People/Teams**	**Tools & Methods**
Planning	• Poor planning and estimation practices • Insufficient reserves for SW	• No generally accepted planning process for software development.; planning is largely dependent on the individual engineer (preparing the plan) • Uniqueness of software not captured in initial stages (functional to deliverable) • SW requirements and design are more volatile & solidify later than hardware in the life cycle. • Don't know how to freeze software requirements the same way we know how to freeze hardware requirements	• SW team not included in early stages of planning • SW not recognized in initial planning	• Poor and constantly changing assumptions and cost estimation methods • Lack of software planning tools • Lack of SW cost metrics.
Requirements & Design	• Lack of good architecture and system partitioning • Systems decisions made without accounting for impact on software	• Subsystem view of spacecraft -- not viewed as important to have a top-level architecture early in the project. • Software design is traditionally done at the subsystem level (based on hardware perspective) • Architectural issues are not sufficiently worked out in Phase A/B • Concurrent development can lead to interface problems due to lack of communication between teams especially when there is schedule compression.	• No awareness or recognition even at the mission & system level that software needs to be addressed. • Don't view architecture as a software intensive process	
Experience & Teaming	• Insufficient software experience among managers and system engineers • Poor teaming between HW/ SW and systems/SW team		• Management and system engineers have limited SW experience • Engineers grew up in a hardware intensive world. • Managers and system engineers do not view software engineers as broad enough. • Lack of software-system engineers • Software culture is underdeveloped at the present	
Testing	• Testbeds; too few, too late, not validated, insufficient capability • Lack of early test planning; lack of functionality,	• Lack of sufficient funding. • Testbeds not listed in WBS; not accountable. • Lack of sufficient schedule or recognition of the importance of testing. • "Big Bang" style testing waits until end to test. • Test documents not in place until late in life cycle	• Lack of education & appreciation of value for testbeds. • Test team not in place until late in life cycle • Integration and SW teams not available to support ATLO	• Dependence on hardware testbeds. • Lack of tools and under utilization of existing tools • Lack of controlled tests and test data
Software Inheritance	• Inherited code did not behave as advertised, was poorly documented, and required more modification than expected	• Lack of software inheritance review process. • Inheritance not distinguished between reusable code and code that has not been designed for that purpose. Inheritance (typically) only reuses the design. • No incentives for projects to develop fully reusable code.	• Many projects fail to bring onboard the original developers when they attempt to inherit software	• Too many advantages of inheritances assumed, esp. cost savings • Cost models don't properly account for COTS, sw inheritance and modification.. • Too often assumed that COTS costs are free

Recommendations in Top Risk Areas Receiving 10 or More Votes

Risk Area	Cost Growth Sources	Recommendations		
		Process	**People/Teams**	**Tools/Methods**
Planning, Estimation & Control	• Poor planning and estimation practices • Insufficient reserves for SW	1. Need a focused end point with clear success criteria 2. Need better tailored risk management plan with appropriate contingencies 3. Allocate larger percentage reserves to software		
Requirements & Design	• Lack of good architecture and system partitioning • Systems decisions made without accounting for impact on software	4. Require that a clear understanding of SW be included as part of NAR approval 5. Need good architecture to define demarcation between HW and SW	6. System Engineers need to understand that the software provides the system level interfaces 7. Do not look at SW as separate item but see as part of an integrated system design	
Experience & Teaming	• Insufficient software experience among Managers and system engineers. • Poor teaming between HW/ SW and systems/SW team		8. Project office needs to have some SW expertise 9. SW team needs to understand system[1] 10. Everyone should have some mission level training to provide end-to-end understanding of the system [1]	
Testing	• Testbeds; too few, too late, not validated, lacked capability • Lack of early test planning; lack of functionality,	11. Testbeds and simulators need to be made a major product deliverable that is completed early in lifecycle	12. Need to have a dedicated integration team and a dedicated test team whose job is it to break the software 13. Require a test engineer be a member of the early planning team and reviews.	
Software Inheritance	• Inherited code did not behave as advertised, was poorly documented, and required more modification than expected	14. Need a software inheritance review	15. For Inheritance people need to come with the software	16. To increase the amount of Inheritance between projects, need to create infrastructure to provide incentives to develop reusable code and to maintain it.

Flight Cost Growth Summary of Key Recommendations

Projects need to have:

- Key personnel with major software experience as part of planning, design and decision making processes
- A system level design that is not primarily hardware oriented but must represent an integrated Hardware-Software design
- Multiple testbeds and simulators available early in life cycle

Policy Recommendations

Recommended JPL Organizational Policy

1. Require all projects have a software system manager with budget authority and responsibility over flight and ground SW and reports directly to the project manager. (The same as the spacecraft and instrument managers.) Among others the software system managers responsibilities include:

Recommended JPL Product Policies

2. Require the development of a system architecture supported by a software architecture that clearly documents an integrated hardware and software design prior to PDR.
3. Require the development of a management plan that addresses software including a risk management plan with reserve and contingency allocations based on estimated risk prior to PDR.
4. Require the development of a test strategy and plan prior to PDR.

Recommended JPL Process Policies

5. Require a Software Inheritance Review similar to the Hardware Inheritance Review (when appropriate) prior to PDR and CDR.
6. Require that software be reviewed at the NAR.
7. Require that the software architectural designs be reviewed at PDR and updated at CDR.
8. Require Risk Management Plan be reviewed at PDR and updated at CDR.
9. Require Test Plans and status be reviewed at PDR and updated at CDR.

Conclusion

Results included in JPL Handbook–

Software Development Principles for Flight Systems: General Principles

Developing a function based approach that combines parametric modeling with quasi-bottom up estimation

Distributed report to senior managers and key JPL personnel

Summary of Initial Recommendations by Risk Area

Risk Area	Summary of Reported Recommendations
Experience & Teaming	•Need project managers & system engineers who understand SW •System engineers need to understand that SW provides the system level interfaces •Project office needs to have some SW expertise •Need to build a team that can work together and communicate •PMs need to be able to identify staffing problems early
Planning	•Need a focused end point with clear success criteria •Need better tailored risk management with contingency plans • Need a plan you can track and hang your hat on based on a complete lifecycle •SW must have an early presence even in pre-Phase A and be part of an integrated plan • Allocate larger reserves to SW •Require that a clear understanding of SW be included as part of NAR approval • Need more detailed planning and tracking of SW similar to HW • When putting together a plan get inputs from everyone and negotiate. Add schedule slack but make sure all manager's know they are accountable • Need to change rules of thumb. E.g., SW development vs. test used to be 50/50 now appears to be 15/85
Requirements & Design	• Must have a development process that deals with evolving reqs & assumes things will break. Early and extensive prototyping Incremental deliveries & evolving documents Isolate interfaces • Identify standardized SW functions and put in HW. •Need good architecture to define demarcation between HW and SW. •Do not look at SW as separate but see as an integrated design • Get a baseline and CM in place so can carefully manage prioritized requirements
Testing	• Need to have many and varied SW test environments • Need to have a dedicated integration and a dedicated test team whose job it is to break the SW • Testbeds and simulators need to be made a major product deliverable that is completed early in lifecycle
Software Inheritance	•Need a software inheritance review •For successful software inheritance, developers need to come with the software
Tools etc.	• Make sure target and development systems are the same • Use design tools with proven record • Get methodology and process in place before purchasing tools • Need good test analysis tools
Staffing	• We need to go outside to get more expertise • Software team needs to understand the system •Plan to over staff SW engineers to deal with turnover •Need a mechanism to hire more SW people without elaborate hiring procedures • Everyone should have some mission level training to provide end-to-end understanding of the system